



**POLITECNICO**  
**MILANO 1863**

SCUOLA DI INGEGNERIA INDUSTRIALE  
E DELL'INFORMAZIONE

# SMBUD Project - Spark

Author(s): **Gabriele Ginestroni**

**Giacomo Gumiero**

**Lorenzo Iovine**

**Nicola Landini**

**Francesco Leone**

Group Number: **10**

Academic Year: 2022-2023



# Contents

<b>Contents</b>	<b>i</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Specification . . . . .	1
1.2 Assumption . . . . .	1
<b>2 Conceptual Model</b>	<b>3</b>
<b>3 Data Structure</b>	<b>5</b>
<b>4 Dataframe Structure</b>	<b>7</b>
4.1 Article Structure . . . . .	7
4.2 Author Structure . . . . .	8
4.3 Venue Structure . . . . .	9
4.4 Dataframe Analysis . . . . .	10
<b>5 Commands and Queries</b>	<b>11</b>
5.1 Commands . . . . .	11
5.1.1 Insert a new author . . . . .	11
5.1.2 Insert a new article . . . . .	12
5.1.3 Insert a new venue . . . . .	13
5.1.4 Insert a new article in his author dataframe . . . . .	13
5.1.5 Update the number of citations of referenced publications . . . . .	13
5.1.6 Deleting an author from the database . . . . .	14
5.2 Queries . . . . .	15
5.2.1 Query 1 - WHERE, JOIN . . . . .	15
5.2.2 Query 2 - WHERE, LIMIT, LIKE . . . . .	15
5.2.3 Query 3 - WHERE, IN, NESTED QUERY . . . . .	16
5.2.4 Query 4 - GROUP BY, JOIN, AS . . . . .	17

5.2.5	Query 5 - WHERE, GROUP BY . . . . .	18
5.2.6	Query 6 - GROUP BY, HAVING, AS . . . . .	18
5.2.7	Query 7 - WHERE, GROUP BY, HAVING, AS . . . . .	19
5.2.8	Query 8 - WHERE, NESTED QUERY, GROUP BY . . . . .	20
5.2.9	Query 9 - WHERE, GROUP BY, HAVING, 1 JOIN . . . . .	21
5.2.10	Query 10 - WHERE, GROUP BY, HAVING, 2 JOINS . . . . .	22
5.2.11	Query 11 - EXTRA . . . . .	23
5.2.12	Query 12 - EXTRA . . . . .	23

# 1 | Introduction

In this chapter will be presented the problem specification and the hypothesis under which the database is implemented.

## 1.1. Problem Specification

This project aims to build a database that handles scientific articles contained in the DBLP bibliography. In this implementation, our work will be focused on *Spark* technology, which is a ditributed computing infrastructure that can process large amount of data in efficient day. To accomplish this we used the PySpark interface that allows us to interact with Apache Spark using python.

## 1.2. Assumption

- An author can't work for more than one organization for the same Publication
- As in the MongoDB implementation, a publication can be published only in one venue
- Venues with the same raw take place in the same city



## 2 | Conceptual Model

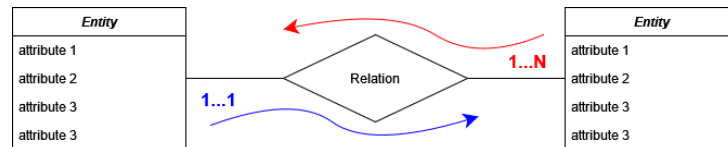


Figure 2.1: ER Diagram Organization

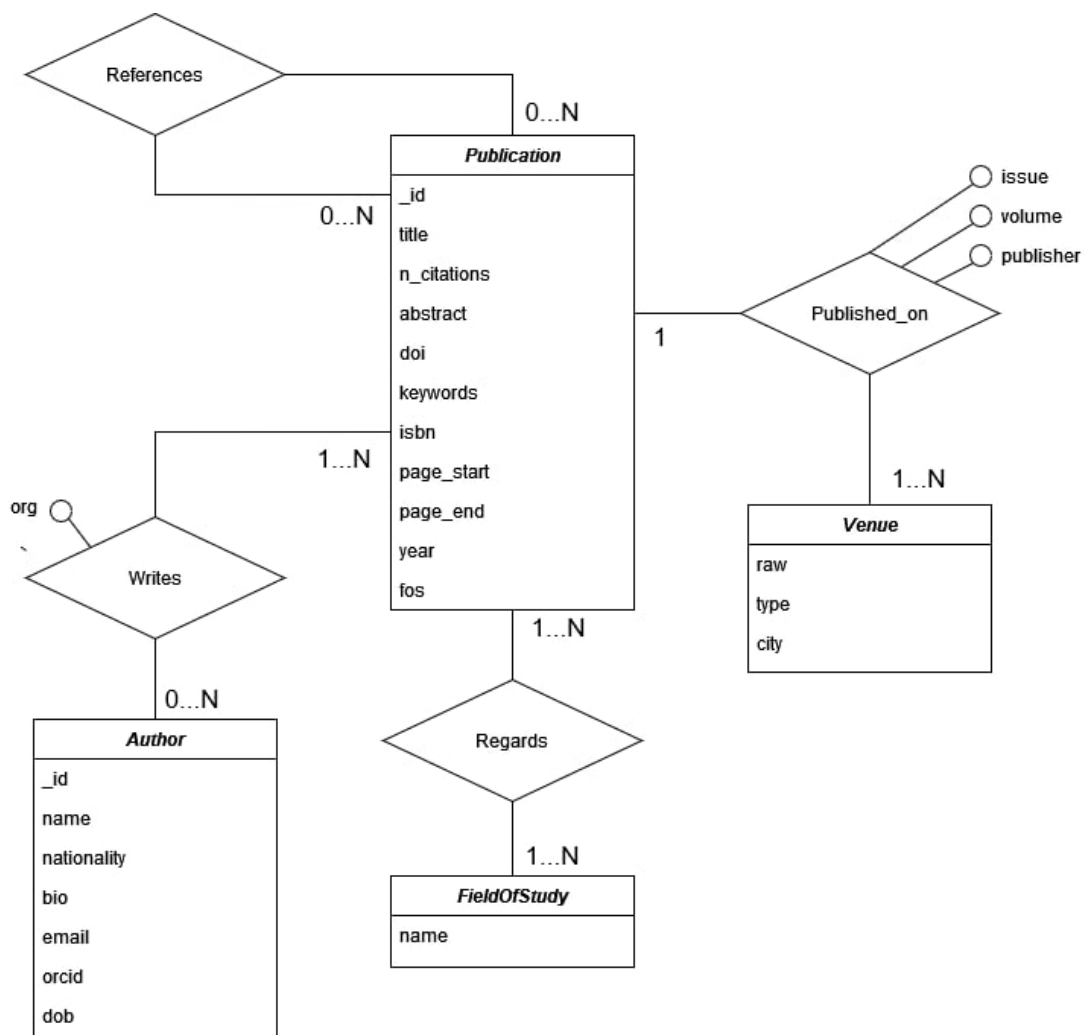


Figure 2.2: Conceptual Model

The conceptual model above contains 6 main entities, that are related to each other through the following relationships:

- **Publication:** represents all the scientific articles. Its attributes are: *\_id*, *title*, *n\_citations*, *abstract*, *doi*, *keywords*, *isbn*, *page\_start*, *page\_end*, *year*, *fos* and its organization will be presented later
- **Author:** it is the one who contributed to a publication. Its attributes are: *\_id*, *name*, *nationality*, *bio*, *email*, *orcid*, *dob* (*date of birth*)
- **Venue:** it is where a publication is published or presented. Its attributes are: *raw*, *type*, *city*
- **FieldOfStudy:** this entity represents the subjects of the publication and its attribute is *name*

Differences with *MongoDB* conceptual model:

- we add field *city* in *Venue* entity representing the location of the venue
- *Chapters* and *images* have been removed

The designed model contains also the following relationships:

- **References:** is the relationship between a *publication* and another *publication* cited by the first one
- **Published\_on:** is the relationship between a *publication* and its *venue*. Its attributes are: *issue*, *volume*, *publisher*
- **Writes:** is the relationship between *author* and *publication* which features the affiliation property. We decided to design it with **org** as an attribute of the relationship, due to the fact that it belongs only to a pair of *author* and *publication* and it represents the institute where the author worked for the publication
- **Regards:** is the relationship between a *publication* and its *fields of study*



## 3 | Data Structure

In this part of the project we used the same two JSONs used in the *MongoDB* implementation. We only removed chapters inside articles and `_id` authors inside articles was renamed as `idAuth`. Also `id` and dates were reconverted in plain text because in *MongoDB* we needed to add them as special entities: `$oid` and `$date`.

This was done via the following lines of script:

- the **first line** is used for the JSON file containing the articles. The script removes `$oid`, to delete chapters field and to rename authors field `id` into `idAuth`
- the **second line** is used for the JSON file containing the authors. The script removes `$oid` and `$date`

```
1 cat dblp_sample_filtered.json | sed -E 's/{"\$oid":(["a-z0-9]+)}\1/g'
  | jq 'del(.[].chapters)' | jq '.[].authors[] |= with_entries(if .key
    == "id" then .key = "idAuth" else . end)' >
  dblp_sample_filtered_spark.json
2
3 cat dblp_sample_reverted_filtered.json | sed -E 's/{"\$oid":(["a-z0
  -9]+)}\1/g' | sed -E 's/{"\$date":(["A-Z0-9:-]+)}\1/g' >
  dblp_sample_reverted_filtered_spark.json
```



# 4 | Dataframe Structure

## 4.1. Article Structure

```

1 root
2 |-- _id: string (nullable = true)
3 |-- title: string (nullable = true)
4 |-- authors: array (nullable = true)
5 |   |-- element: struct (containsNull = true)
6 |   |   |-- idAuth: string (nullable = true)
7 |   |   |-- org: string (nullable = true)
8 |-- n_citation: integer (nullable = true)
9 |-- abstract: string (nullable = true)
10 |-- doi: string (nullable = true)
11 |-- keywords: array (nullable = true)
12 |   |-- element: string (containsNull = true)
13 |-- isbn: string (nullable = true)
14 |-- page_start: string (nullable = true)
15 |-- page_end: string (nullable = true)
16 |-- year: integer (nullable = true)
17 |-- fos: array (nullable = true)
18 |   |-- element: string (containsNull = true)
19 |-- references: array (nullable = true)
20 |   |-- element: string (containsNull = true)
21 |-- issue: string (nullable = true)
22 |-- volume: string (nullable = true)
23 |-- publisher: string (nullable = true)
24 |-- venue_raw: string (nullable = true)

```

The structure just shown represents an *Article*; its attributes are:

- **\_id** is the identifier of a publication.
- **title** represents the title of the publication.
- **authors** is an array that contains: **idAuth** of the authors of the article and the **org** field which represent the affiliation.

- **n\_citation** is the number of times that the publication has been mentioned.
- **abstract** is a string containing a brief summary of the contents of the paper.
- **doi** Digital Object Identifier is a persistent and standardized identifier.
- **keywords** is an array containing keywords of the publication.
- **isbn** is an identification code of the venue of the publication.
- **page\_start** defines the starting page of the publication.
- **page\_end** defines the last page of the publication.
- **year** represents the year of publication.
- **fos** is an array containing the fields of study of the publication.
- **references** set of ObjectIds of the referenced articles.
- **issue** refers to how many times a periodical has been published during that year.
- **volume** is the volume of the venue in which the article has been published.
- **publisher** is the name of the publisher of the article.
- **venue\_raw** is the name or the abbreviation of the venue (regardless the year, issue or volume) in which the publication was presented.

We moved **issue**, **volume** and **publisher** to the *Publication* structure because in this implementation we have a new collection for the *Venue*. This has been done because, as in previous projects, we decided to aggregate the venues with respect to the field **raw**.

## 4.2. Author Structure

```

1 root
2 |-- _id: string (nullable = true)
3 |-- name: string (nullable = true)
4 |-- nationality: string (nullable = true)
5 |-- articles: array (nullable = true)
6 |   |-- element: string (containsNull = true)
7 |-- bio: string (nullable = true)
8 |-- email: string (nullable = true)
9 |-- orcid: string (nullable = true)
10 |-- dob: timestamp (nullable = true)

```

The structure just shown represents an *Author*; its attributes are:

- **\_id** is the identifier of an author.
- **name** is the name of the author.
- **nationality** is the nationality of the author.
- **articles** is a set of articles identifier of the publications of the author.
- **bio** is a string that describes the author.
- **email** is the email address of the author.
- **orcid** Open Researcher and Contributor ID is a unique identifier for authors of scientific articles.
- **dob** is the birth date of the author.

### 4.3. Venue Structure

```

1 root
2 |-- raw: string (nullable = true)
3 |-- type: integer (nullable = true)
4 |-- artIds: array (nullable = false)
5 |     |-- element: string (containsNull = false)
6 |-- city: string (nullable = true)

```

The structure just shown represents a *Venue*. This dataframe was obtained extracting venues fields from the Articles dataframe already imported. *Venue* attribute are the following:

- **raw** is the name or the abbreviation of the venue (regardless the year, issue or volume) in which the publication was presented.
- **type** indicates the type of the publication.
- **artIds** is a set of articles identifier associated to the venue.
- **city** represents the location of the venue an it is randomly populated.

Note that *artIds*, *city* where not present in the Article dataframe so have been generated during the creation of the venue collection.

## 4.4. Dataframe Analysis

These are the number of rows in our dataframes

```
print("df_articles rows:" + str(df_articles.count()))  
print("df_authors rows:" + str(df_authors.count()))  
print("df_articles venues:" + str(df_venues.count()))
```

✓ 3.8s

```
df_articles rows:6381  
df_authors rows:17025  
df_articles venues:3001
```



### 5.1.2. Insert a new article

Assuming it is not present in the dataset, we created row with a new article written by the author created in section 5.1.1. In order to set the authors, we instantiated an array `new_authors`.

```

1 new_authors = [Row("638db170ae9ea0d19fad7a79", "Politecnico di Milano")
2               , Row("638db170ae9ea0d19fad7a7a", "
3                   Politecnico di Milano")]
4
5 new_article = Row(
6     _id="638db237d794b76f45c77916",
7     title="An extensive study of C-SMOTE, a Continuous Synthetic
8           Minority Oversampling Technique for
9           Evolving Data Streams",
10    authors=new_authors,
11    n_citation=3,
12    abstract = "Streaming Machine Learning (SML) studies algorithms that
13               update their models,\
14               given an unbounded and often non-stationary flow of data
15               performing a single pass. Online \
16               class imbalance learning is a branch of SML that combines the
17               challenges of both class imbalance\
18               and concept drift. In this paper, we investigate the binary
19               classification problem by
20               rebalancing\
21               an imbalanced stream of data in the presence of concept drift,
22               accessing one sample at a time.",
23    doi="10.1016/j.eswa.2022.116630",
24    keywords=["Evolving Data Stream","Streaming","Concept drift","
25              Balancing"],
26    isbn="123-4-567-89012-3",
27    page_start="39",
28    page_end="46",
29    year=2022,
30    fos=["Computer Science","Stream Reasoning","Big Data"],
31    references=["53e99fe4b7602d97028bf743","53e99fddb7602d97028bc085"],
32    issue="1",
33    volume="196",
34    publisher="Elsevier",
35    venue_raw="ESA"
36 )

```



```
27 df_articles = df_articles.union(spark.createDataFrame([new_article]))
```

### 5.1.3. Insert a new venue

Assuming it is not present in the dataset, we created a new row with the values for a new venue *ESA* hosted in *Montreal*.

**Note:** in field `artIds` we set the article created in section 5.1.2.

```
1 new_venue = Row(
2     raw="ESA",
3     type=1,
4     artIds=["638db237d794b76f45c77916"],
5     city="Montreal"
6 )
7
8 df_venues = df_venues.union(spark.createDataFrame([new_venue]))
```

### 5.1.4. Insert a new article in his author dataframe

Through this command we inserted the article created in section 5.1.2 to its authors. In order to do that, we selected the authors through the ids and we add the article id to their field `articles`.

**Note:** one of the author is the one created in section 5.1.1.

```
1 df_authors = df_authors.withColumn(
2     "articles",
3     f.when(f.col("_id") == "638db170ae9ea0d19fad7a79",
4         f.array_union(df_authors.articles, f.array(f.lit("
5             638db237d794b76f45c77916"))))\
6         .when(f.col("_id") == "638db170ae9ea0d19fad7a7a",
7             f.array_union(df_authors.articles, f.array(f.lit("
8                 638db237d794b76f45c77916"))))
9         .otherwise(f.col("articles"))
10 )
```

### 5.1.5. Update the number of citations of referenced publications

Through the following snippet of code is possible to increment the `n_citations` field of the *Publications* referenced by the article created in section 5.1.2.

**Note:** field `n_citations` is updated for both the referenced articles.

```
1 df_articles = df_articles.withColumn(  
2     "n_citation",  
3     f.when(f.col("_id") == "53e99fe4b7602d97028bf743",  
4         df_articles.n_citation+1) \  
5     .when(f.col("_id") == "53e99fddb7602d97028bc085",  
6         df_articles.n_citation+1)  
7     .otherwise(f.col("n_citation"))  
8 )
```

### 5.1.6. Deleting an author from the database

Through the following snippet of code is possible to delete an author from the database. In order to do that we started from filtering on the identifier of the author to be removed and we deleted it.

```
1 df_authors.filter(f.col("_id") == "638db170ae9ea0d19fad7a79").show()  
2 df_authors = df_authors.filter(f.col("_id") != "638db170ae9ea0d19fad7a79"  
                                ")
```

## 5.2. Queries

We have identified the following queries in order to show the system's basic functionalities. In the following sections title we wrote the basic requirements for every query, that, for ease of read, are represented as SQL clauses.

### 5.2.1. Query 1 - WHERE, JOIN

This query returns the type of the venue of an article with the following title: *"Locality Sensitive Outlier Detection: A ranking driven approach"*.

**Description:** starting from the articles dataframe, a join is performed with the venues dataframe on the article's `venue_raw` field. After that, we filter the articles with the given title. Finally, we project over title,venue raw and venue type.

```

1 df_articles.join(df_venues, df_articles.venue_raw == df_venues.raw, "
                                inner")\
2     .filter(f.col("title") == "Locality Sensitive Outlier
                                Detection: A ranking driven approach
                                ") \
3     .select("title", "raw", "type")\
4     .show(truncate=False)

```

```

+-----+-----+-----+
|title                                     |raw |type|
+-----+-----+-----+
|Locality Sensitive Outlier Detection: A ranking driven approach|ICDE|0  |
+-----+-----+-----+

```

### 5.2.2. Query 2 - WHERE, LIMIT, LIKE

This query returns the articles whose title string contains *"Machine Learning"*.

**Description:** we filter the articles whose title contains "Machine Learning" using the like operator. Results are then limited to 3 tuples and projected over the article title.

```

1 df_articles.filter(f.col("title").like("%Machine Learning%"))\
2     .limit(3)\
3     .select("title")\
4     .show(truncate=False)

```

```
+-----+
|title|
+-----+
|Editorial: The Terminology of Machine Learning|
|Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005|
|Medical Expert Evaluation of Machine Learning Results for a Coronary Heart Disease Database|
+-----+
```

### 5.2.3. Query 3 - WHERE, IN, NESTED QUERY

This query finds authors that has the same nationality of at least one of the authors of *"Locality Sensitive Outlier Detection: A ranking driven approach"* article.

**Description:** this query has been splitted in 2 queries:

- *First query:* articles are filtered to find the article with the given title. After that, the authors array is exploded to perform a join on its `idAuth` field with the authors dataframe. Finally, nationalities of the article's authors are collected into a list using the `collect_set`.  
`collect_set`, as the name suggests, discards duplicates, so the final list is a set of nationalities.
- *Second query:* starting from the authors' dataframe, we filtered all the authors whose nationality is present inside the list created with the previous query.

```
1 nationalities_list = df_articles.filter(f.col("title") == "Locality
                                     Sensitive Outlier Detection: A
                                     ranking driven approach")\
2                                     .select(f.explode(df_articles.authors.
                                     idAuth).alias("idAuth"))\
3                                     .join(df_authors, on=f.col("idAuth") ==
                                     df_authors._id)\
4                                     .select("nationality")\
5                                     .agg(f.collect_set("nationality")).
                                     collect()[0][0]
6
7 df_authors.filter(f.col("nationality")\
8                 .isin(nationalities_list))\
9                 .select("name","nationality")\
10                .show(truncate=False)
```

name	nationality
Ye Wang	dk
Srinivasan Parthasarathy	jp
Shirish Tatikonda	gr
Moshe Zukerman	jp
Michael Wiegand	jp
GeunSik Jo	jp
Carla Achury	gr
Kong-Aik Lee	jp
Shahram Shah-Heydari	gr
Wenfang Tan	dk
Ayoub Alsarhan	gr
Anjali Agarwal	jp
David Haccoun	jp
Silvio Macedo	dk
John Wan Tung Lee	gr
Geoff Holmes	dk
Zornitsa Kozareva	jp
Peter Murray-Rust	jp
Rajkumar Buyya	jp
Srikumar Venugopal	jp

only showing top 20 rows

#### 5.2.4. Query 4 - GROUP BY, JOIN, AS

This query finds the 3 most frequent keywords of articles written by italian authors.

**Description:** starting from the authors dataframe, we keep only italian authors and explode the articles field, renaming the new obtained field to `articles`. After that, duplicates are discarded.

In the second part of the query, we load the full articles's rows using a join. Then, keywords array is exploded. Keywords are grouped and counted. The groups are finally sorted and limited to show the top 3 keywords.

```

1 df_italian = df_authors.filter(f.col("nationality") == "it")\
2     .select(f.explode("articles")).withColumnRenamed(
3         "col", "articles")\
4     .distinct()
5 df_keywords = df_italian.join(df_articles, df_italian.articles ==
6     df_articles._id, "inner")\
7     .select("articles", f.explode("keywords"))\
8     withColumnRenamed("col", "keywords")\
9     .groupby("keywords")\
10    .agg(f.count("keywords").alias("n_occurences"))\
    .sort("n_occurences", ascending=False)\
    .limit(3).show()

```

keywords	n_occurences
data mining	27
computer science	22
internet	17

### 5.2.5. Query 5 - WHERE, GROUP BY

This query finds the cities with more than 65 venues.

**Description:** the venues dataframe is grouped with respect to the city to perform the count. After that, we keep only cities with more than 65 venues and sort the result in descending order.

```

1 df_venues\
2   .groupby("city")\
3   .count()\
4   .filter(f.col("count") > 65)\
5   .sort("count", ascending=False).show()

```

city	count
Paris	78
Istanbul	72
Vienna	68
Riga	68
Tbilisi	66

### 5.2.6. Query 6 - GROUP BY, HAVING, AS

This query finds the field of studies that appears more than 15 times.

**Description:** We use the explode function to convert the fos array into multiple rows, then we rename the resulting column to fos, group by fos and count the number of occurrences.

After that, we keep rows with more than 15 occurrences, sort the remaining rows in descending order based on the number of occurrences, and show the top rows.

```

1 df_articles\
2   .select("_id", "title", f.explode("fos")).withColumnRenamed("col", "
                                     fos")\

```

```

3     .groupby("fos")\
4     .agg(f.count("fos").alias("n_occurencies"))\
5     .filter(f.col("n_occurencies") > 15)\
6     .sort("n_occurencies", ascending=False)\
7     .show(truncate=False)

```

fos	n_occurencies
Computer science	3988
Artificial intelligence	1246
Mathematics	1194
Algorithm	575
Computer network	452
Computer vision	395
Distributed computing	388
Engineering	374
Pattern recognition	333
Data mining	327
Theoretical computer science	326
Discrete mathematics	294
Mathematical optimization	293
World Wide Web	264
Machine learning	263
Combinatorics	239
Control theory	227
Information retrieval	222
Programming language	217
Knowledge management	203

only showing top 20 rows

### 5.2.7. Query 7 - WHERE, GROUP BY, HAVING, AS

This query finds all the volumes with at least 5 articles in the dataset, published after 2000.

**Description:** This query filters the articles in the articles dataframe to only those published after the year 2000, then groups the remaining articles by `venue_raw` and `volume`, counts the number of articles per group, filters the groups to only those with more than 4 articles, and finally displays the results.

```

1 df_articles\
2     .filter(f.col("year") > 2000)\
3     .groupby("venue_raw", "volume")\
4     .agg(f.count("volume").alias("num_articles"))\
5     .filter(f.col("num_articles") > 4)\
6     .show(truncate = False)

```

venue_raw	volume	num_articles
Applied Mathematics and Computation	218	5
Pattern Recognition	45	5
Expert Syst. Appl.	39	5
Applied Mathematics and Computation	217	5
IEICE Transactions	97-A	5
Expert Syst. Appl.	37	5

### 5.2.8. Query 8 - WHERE, NESTED QUERY, GROUP BY

The following query is divided in two queries:

- 8a. find the venue with highest number of articles
- 8b. find the number of articles published per year on that venue

**Description:** the basic functionalities of the two queries are the following

- the *first query* selects the top venue from the venue dataframe based on the size of the `artIds` attribute
- the *second query* filters articles with the selected `venue_raw`, groups the articles by year, and counts the number of articles in each group. Finally, it displays the results projecting over `top_venue`, `year` and `articles_count`

```

1 top_venue = df_venues\
2     .select("raw",f.size("artIds").alias("count"))\
3     .orderBy("count",ascending = False)\
4     .limit(1)
5
6 df_articles_year = df_articles\
7     .filter(f.col("venue_raw") == top_venue.collect
8           () [0] [0])\
9     .groupBy("year")\
10    .count()\
11    .orderBy("count",ascending=False)\
12    .select(f.lit(top_venue.collect() [0] [0]).alias("
    VenueRAW"),"year",f.col("count").
    alias("articles_count"))\
    .show(truncate=False)

```



VenueRAW	year	articles_count
Clinical Orthopaedics and Related Research	2010	11
Clinical Orthopaedics and Related Research	2011	8
Clinical Orthopaedics and Related Research	2009	7
Clinical Orthopaedics and Related Research	2008	5
Clinical Orthopaedics and Related Research	2007	3
Clinical Orthopaedics and Related Research	2000	2
Clinical Orthopaedics and Related Research	2012	1
Clinical Orthopaedics and Related Research	2006	1
Clinical Orthopaedics and Related Research	2013	1
Clinical Orthopaedics and Related Research	2005	1
Clinical Orthopaedics and Related Research	2001	1

```

9      .select("title","different_nationalities
      ",f.sort_array("nationalities_list")
      .alias("nationalities_list"))\
10     .show(truncate=False)

```

title	different_nationalities	nationalities_list
Being user-oriented: Convergences, divergences, and the potentials for systematic dialogue between disciplines and between researchers, designers, and providers	17	[[de, de, de, fr, gr, hu, it, jp, nl, no, pl, pt, ro, ru, se, tr, us]]
A Low-Power Single-Weight-Combiner 882.11nbg SoC in 0.13 μm CMOS for Embedded Applications Utilizing An Area and Power Efficient Cartesian Phase Shifter and Mixer Circuit	17	[[de, de, es, fr, gr, hu, it, jp, nl, no, pl, ro, ru, se, tr, uk, us]]
A 70b/s/pin 60GDS SDRAM with 2.5ns bank-to-bank active time and no bank-group restriction	18	[[de, de, fr, gr, hu, it, jp, no, pl, pt, ro, ru, se, tr, uk]]
Cluster Analysis and Decision Trees of MR Imaging in Patients Suffering Alzheimer's	18	[[de, de, es, gr, jp, nl, no, pt, ro, ru, se, tr, uk, us]]

### 5.2.10. Query 10 - WHERE, GROUP BY, HAVING, 2 JOINS

This query finds all the authors that published on more than 2 Journals.

**Description:** starting from the authors dataframe, we explode the articles array, creating a new field named `article`. After that, we join the results with the articles collection and then with the venues collection. Then, we filter the results to keep only articles written on journals (type 1) and group by the author id. Finally, we count the number of distinct venues in each group (collecting in a list all the venues of the group), and keep only the groups with more than 2 venues.

```

1 df_exploded_authors = df_authors.alias("auth")\
2     .select("auth._id","auth.name", f.explode("auth.
3         articles").alias("article"))\
4     .join(df_articles.alias("art"), on=f.col("
5         article") == df_articles._id)\
6     .select("auth._id","auth.name","art._id","art.
7         venue_raw")\
8     .join(df_venues.alias("ven"), on=f.col("
9         venue_raw") == df_venues.raw)\
10    .filter(f.col("type") == 1)\
    .groupBy("auth._id")\
    .agg(f.first("name").alias("name"),f.
        countDistinct("raw").alias("
        venue_count"),f.concat_ws(" - ",f.
        collect_set("raw")).alias("
        venues_list"))\
    .filter(f.col("venue_count") > 2)\
    .orderBy("venue_count", ascending=False).show(3,
        truncate=False)

```

_id	name	venue_count	venues_list
54055740dabfae44f0803fbb	Naohiro Ishii	3	Las Vegas, NV - Honolulu, HI - International Journal on Artificial Intelligence Tools

### 5.2.11. Query 11 - EXTRA

This query returns all the articles written by authors whose names combined have all 26 letters of the alphabet.

**Description:** The query starts with exploding articles for each author. Then, the grouping combined with the collect retrieves, for each article, the list of its authors, then several operation are applied on this list in order to obtain the different letters that are present in the list of authors. After that, a filter to keep only the ones that have all the 26 letters of the alphabet in it is applied, and the result is joined with articles to obtain the title. Finally, a projection is used to display the title and the list of authors in alphabetical order.

```

1 df_authors.select("name", f.explode("articles").alias("idArt")) \
2     .groupBy("idArt") \
3     .agg(f.collect_set("name").alias("authorsList")) \
4     .select("idArt", "authorsList", (f.size(f.array_distinct(f.
5                                     split(f.regexp_replace(f.lower(f.
6                                     concat_ws(" ", "authorsList")), "[^a-
7                                     z]", ""), ""))) - 1).alias("
8                                     differentLetters")) \
9
10    .filter(f.col("differentLetters") == 26) \
11    .join(df_articles, on=f.col("idArt") == df_articles._id) \
12    .select("title", f.concat_ws(" ", f.sort_array("authorsList")
13                                     ).alias("authorsList"), "
14                                     differentLetters") \
15
16    .show(truncate=False)

```

title	authorsList	differentLetters
Design principles for developing stream processing applications	Rugra Gupta, Chitra Venkatesan, Suresh S. Iyengar, Henrique Andrade, Jeffrey David Harris, John Cox, Olivier Verschuer, Paul Jones, William Sanchez	26
Building an information retrieval test collection for spontaneous conversational speech	Shoubo Kanabhai, Sijbert Sijpe, David S. Borwein, Douglas A. Gelf, S. Craig Murray, James Hayfield, Jianlong Wang, Vilijko Stancov, Martin Frenn, Samuel Guttman, Stephanie Strassel, Xianli Huang	26

### 5.2.12. Query 12 - EXTRA

This query returns all articles written in affiliation with *Politecnico of Milano*.

**Description:** the query starts by exploding the authors array field in the article, creating

the new `affiliation` attribute. Articles that contain at least one of the desired organization (the same article could be written in collaboration with different universities) are kept. Then, a join with the authors collection is executed to retrieve the name of the author.

```
1 df_articles\
2   .select("title",f.explode("authors").alias("affiliation"))\
3   .filter(f.col("affiliation.org").like("%Poli%Mil%"))\
4   .join(df_authors, on=f.col("affiliation.idAuth") == df_authors._id)
5   \
6   .select("title", "name", "affiliation.org") \
7   .orderBy("title","name")\
   .show(truncate=False)
```

title	name	org
"The Fire and The Mountain": tangible and social interaction in a museum exhibition for children	Franca Garzotto	Politecnico di Milano, Milano, Italy
"The Fire and The Mountain": tangible and social interaction in a museum exhibition for children	Francesca Rizzo	Politecnico di Milano, Milano, Italy
A Logical Model for Agent Communication Languages	Marco Colombetti	Politecnico di Milano Milano, Italy University of Lugano Lugano, Switzerland
A Logical Model for Agent Communication Languages	Mario Verdicchio	Department of Electronics and Information Politecnico di Milano Milano, Italy
A posteriori dual-sized adaptive finite element error control for Lamé and Stokes equations	Riccardo Sacco	Dipartimento di Matematica "P. Dirichlet", Politecnico di Milano, via Bonardi 9, 20133, Milano, Rocquencourt, Italy
Coordinated cutting plane generation via multi-objective separation.	Eduardo Amaldi	Dipartimento di Elettronica ed Informazione, Politecnico di Milano, Milano, Italy
Coordinated cutting plane generation via multi-objective separation.	Stefano Coniglio	Dipartimento di Elettronica ed Informazione, Politecnico di Milano, Milano, Italy
Coordinated cutting plane generation via multi-objective separation.	Stefano Gualandri	Dipartimento di Elettronica ed Informazione, Politecnico di Milano, Milano, Italy
Hierarchy-based mining of association rules in data warehouses	Giuseppe Psaila	Politecnico di Milano, Dipartimento di Elettronica e Informazione, Piazza L. Da Vinci, 32, I-20133 Milano, Italy
Hierarchy-based mining of association rules in data warehouses	Pier Luca Lanzi	Politecnico di Milano, Dipartimento di Elettronica e Informazione, Piazza L. Da Vinci, 32, I-20133 Milano, Italy
ICT and mobile health to improve clinical process delivery. a research project for therapy management process innovation.	Nicola Restifo	Fdm Politecn Milano, Milan, Italy
ICT and mobile health to improve clinical process delivery. a research project for therapy management process innovation.	Paolo Lucarelli	Fdm Politecn Milano, Milan, Italy
ICT and mobile health to improve clinical process delivery. a research project for therapy management process innovation.	Roberta Facchini	Fdm Politecn Milano, Milan, Italy
Live goals for adaptive service compositions	Liliana Pasquale	Politecnico di Milano, Milano, Italy
Live goals for adaptive service compositions	Luciano Saresi	Politecnico di Milano, Milano, Italy
Parallel conjugate gradient with Schwarz preconditioner applied to fluid dynamics problems	A. Quarteroni	Politecnico di Milano, P.zza Leonardo da Vinci, 32, I-20133 Milano, Italy
Refining and Compressing Abstract Model Checking	Elena Quintarelli	Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy
Risk analysis of underground infrastructures in urban areas	Massimiliano De Ambroggi	Department of Management, Economics and Industrial Engineering, Politecnico di Milano, Piazza Leonardo da Vinci 32, Milan 20132, Italy
Risk analysis of underground infrastructures in urban areas	Ottavio Grande	Department of Management, Economics and Industrial Engineering, Politecnico di Milano, Piazza Leonardo da Vinci 32, Milan 20132, Italy
Risk analysis of underground infrastructures in urban areas	Paolo Trucco	Department of Management, Economics and Industrial Engineering, Politecnico di Milano, Piazza Leonardo da Vinci 32, Milan 20132, Italy

only showing top 28 rows